Local Memory With Indicator Bits to Support Concurrent DMA and CPU Access

Serge Lasserre

This application claims priority to European Application Serial No. 00402331.3, filed August 21, 2000 (TI-31366EU) and to European Application Serial No. 01400687.8, filed March 15, 2001 (TI-31354EU). US Patent Application Serial No. _____ (TI-31366US) is incorporated herein by reference.

Field of the Invention

This invention generally relates to microprocessors, and more specifically to improvements in cache memory and access circuits, systems, and methods of making.

Background

Microprocessors are general purpose processors which provide high instruction throughputs in order to execute software running thereon, and can have a wide range of processing requirements depending on the particular software applications involved. A cache architecture is often used to increase the speed of retrieving information from a main memory. A cache memory is a high speed memory that is situated between the processing core of a processing device and the main memory. The main memory is generally much larger than the cache, but also significantly slower. Each time the processing core requests information from the main memory, the cache controller checks the cache memory to determine whether the address being accessed is currently in the cache memory. If so, the information

is retrieved from the faster cache memory instead of the slower main memory to service the request. If the information is not in the cache, the main memory is accessed, and the cache memory is updated with the information.

Many different types of processors are known, of which microprocessors are but one example. For example, Digital Signal Processors (DSPs) are widely used, in particular for specific applications, such as mobile processing applications. DSPs are typically configured to optimize the performance of the applications concerned and to achieve this they employ more specialized execution units and instruction sets. Particularly in applications such as mobile telecommunications, but not exclusively, it is desirable to provide ever increasing DSP performance while keeping power consumption as low as possible.

To further improve performance of a digital system, two or more processors can be interconnected. For example, a DSP may be interconnected with a general purpose processor in a digital system. The DSP performs numeric intensive signal processing algorithms while the general purpose processor manages overall control flow. The two processors communicate and transfer data for signal processing via shared memory. A direct memory access (DMA) controller is often associated with a processor in order to take over the burden of transferring blocks of data from one memory or peripheral resource to another and to thereby improve the performance of the processor.

Summary of the Invention

Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. In accordance with a first aspect of the invention, there is provided a digital system having at least one processor, with an associated multi-segment local memory circuit. Within the local memory, a data array is arranged as a plurality of lines each having one or more segments. Each of the segments has a corresponding indicator bit within a set of indicator bits for indicating a condition of the corresponding segment. Direct memory access (DMA) circuitry is connected to the local memory and is operable to transfer data to a selectable portion of segments in the local memory from a selectable region of a second memory. The DMA circuitry also manipulates a portion of the indicator bits that correspond to this portion of segments. The local memory is not a cache in that it does not overlay a portion of secondary memory, but is instead an independent portion of memory that occupies it's own portion of the address space.

In an embodiment of the invention, the indicator bits are valid bits operable to indicate that a corresponding segment contains valid data. The DMA circuitry is operable to set to a valid state the selected portion of indicator bits corresponding to the selectable portion of segments.

In another embodiment, miss detection circuitry is connected to the plurality of valid bits. The miss detection circuitry has a miss signal for indicating when a miss is detected in response to a request from the processor to a segment in the local memory. The processor stalls in response to the miss signal until the DMA circuitry sets a valid bit corresponding to the requested segment to a valid state.

In another embodiment of the invention, the indicator bits are dirty bits operable to indicate that a corresponding segment contains dirty data. The DMA circuitry is operable to transfer data from a selectable portion of segments of the local memory to a selectable region of the second memory in accordance with a corresponding portion of dirty bits, such that only segments within the selectable

portion of segments whose corresponding dirty bits are in a dirty state are transferred.

An aspect of the invention is a method of operating a digital system having a processor and a local memory. The local memory is organized as a plurality of segments. A plurality of indicator bits is associated with the plurality of segments such that each segment has at least one corresponding indicator bit. An indicator bit associated with a segment in the local memory is set to a first state in response to a direct memory access (DMA) transfer of a data value or an instruction from a selectable location in a second memory to the segment in the local memory.

Brief Description of the Drawings

Particular embodiments in accordance with the invention will now be described, by way of example only, and with reference to the accompanying drawings in which like reference signs are used to denote like parts and in which the Figures relate to the digital system of Figure 1 and in which:

Figure 1 is a block diagram of a digital system that includes an embodiment of the present invention in a megacell core having multiple processor cores;

Figure 2A and 2B together is a more detailed block diagram of the megacell core of Figure 1;

Figure 3 is a block diagram illustrating a shared translation lookaside buffer (TLB) and several associated micro-TLBs (μTLB) included in the megacell of Figure 2;

Figure 4 is a block diagram illustrating a configurable cache that is included in the megacell of Figure 1 that has a cache and a RAM-set;

Figure 5 is a flow chart illustrating operation of the hit/miss logic of the configurable cache of Figure 4;

Figure 6 is an illustration of loading a single line into the RAM-set of Figure 4;

Figure 7 is an illustration of loading a block of lines into the RAM-set of Figure 4;

Figure 8 is an illustration of interrupting a block load of the RAM-set according to Figure 7 in order to load a single line within the block;

Figure 9 is a flow diagram illustrating an interruptible block operation on the memory circuitry of Figure 4;

Figure 10 is a block diagram of the cache of Figure 7 illustrating data flow for interruptible block prefetch and clean functions in the RAM-set portion;

Figure 11 illustrates operation of the cache of Figure 4 in which a block of lines is cleaned or flushed in the set associative portion;

Figure 12 is a flow diagram illustrating cleaning of a line or block of lines of the cache of Figure 4;

Figure 13 is a block diagram of an embodiment of a local memory similar to the RAM-set memory of Figure 7 that uses of a set of indicator bits to support concurrent CPU and DMA access;

Figure 13B is a schematic for an address calculation circuit for the DMA system of Figure 13A;

Figure 14 is a schematic illustration of operation of an alternative embodiment of the local memory of Figure 13A;

Figure 15 is a schematic illustration of operation of the indicator bits of the local memory of Figure 13A during system initialization;

Figure 16A and Figure 16B are schematic illustrations of operation of the local memory of Figure 13A with mode circuitry controlling DMA transfers according to dirty bits;

Figure 17A and Figure 17B are schematic illustrations of operation of the local memory of Figure 13A with mode circuitry controlling setting of dirty bits by a DMA transfer;

Figure 18A and Figure 18B are schematic illustrations of operation of the local memory of Figure 13A with mode circuitry controlling DMA transfers according to dirty bits; and

Figure 19 is a representation of a telecommunications device incorporating an embodiment of the present invention.

Corresponding numerals and symbols in the different figures and tables refer to corresponding parts unless otherwise indicated.

Detailed Description of Embodiments of the Invention

[01] Although the invention finds particular application to Digital Signal Processors (DSPs), implemented, for example, in an Application Specific Integrated Circuit (ASIC), it also finds application to other forms of processors. An ASIC may contain one or more megacells which each include custom designed functional circuits combined with pre-designed functional circuits provided by a design library.

[02] Figure 1 is a block diagram of a digital system that includes an embodiment of the present invention in a megacell core 100 having multiple processor cores. In the interest of clarity, Figure 1 only shows those portions of megacell 100 that are relevant to an understanding of an embodiment of the present invention. Details of general construction for DSPs are well known, and may be found readily elsewhere. For example, U.S. Patent 5,072,418 issued to Frederick Boutaud, et al, describes a DSP in detail. U.S. Patent 5,329,471 issued to Gary Swoboda, et al, describes in detail how to test and emulate a DSP. Details of portions of megacell 100 relevant to an embodiment of the present invention are explained in sufficient detail herein below, so as to enable one of ordinary skill in the microprocessor art to make and use the invention.

[03] Referring again to Figure 1, megacell 100 includes a control processor (MPU) 102 with a 32-bit core 103 and a digital signal processor (DSP) 104 with a DSP core 105 that share a block of memory 113 and a cache 114, that are referred to as a level two (L2) memory subsystem 112. A traffic control block 110 receives transfer requests from a memory access node in a host processor 120, requests from control processor 102, and transfer requests from a memory access node in DSP 104. The traffic control block interleaves these requests and presents them to the shared memory and cache. Shared peripherals 116 are also accessed via the traffic control block. A direct memory access controller 106 can transfer data between an external source such as off-chip memory 132 or on-chip memory 134 and the shared memory. Various application specific processors or hardware accelerators 108 can also be

included within the megacell as required for various applications and interact with the DSP and MPU via the traffic control block.

[04] External to the megacell, a level three (L3) control block 130 is connected to receive memory requests from internal traffic control block 110 in response to explicit requests from the DSP or MPU, or from misses in shared cache 114. Off chip external memory 132 and/or on-chip memory 134 is connected to system traffic controller 130; these are referred to as L3 memory subsystems. A frame buffer 136 and a display device 138 are connected to the system traffic controller to receive data for displaying graphical images. Host processor 120 interacts with the resources on the megacell via system traffic controller 130. A host interface connected to traffic controller 130 allows access by host 120 to megacell 100 internal and external memories. A set of private peripherals 140 are connected to the DSP, while another set of private peripherals 142 are connected to the MPU.

[05] Figure 2, comprised of Figure 2A Figure 2B together, is a more detailed block diagram of the megacell core of Figure 1. DSP 104 includes a configurable cache 203 that is configured as a local memory 200 and data cache 202, and a configurable cache 204 that is configured as instruction cache 206 and a RAM-set 208, which are referred to as level one (L1) memory subsystems. The DSP is connected to the traffic controller via an L2 interface 210 that also includes a translation look-aside buffer (TLB) 212. A DMA circuit 214 is also included within the DSP. Individual micro TLBs (μTLB) 216-218 are associated with the DMA circuit, data cache and instruction cache, respectively.

[06] Similarly, MPU 102 includes a configurable cache 223 that is configured as a local memory 220 and data cache 222, and a configurable cache 224 that is configured as instruction cache 226 and a RAM-set 228, again referred to as L1 memory subsystems. The MPU is connected to traffic controller 110 via an L2 interface 230 that also includes a TLB 232. A DMA circuit 234 is also included

within the MPU. Individual micro TLBs (μTLB) 236-238 are associated with the DMA circuit, data cache and instruction cache, respectively.

[07] L2 traffic controller 110 includes a TLB 240 and a micro-TLB (μTLB) 242 that is associated with system DMA block 106. Similarly, L3 traffic controller 130 includes a μTLB controllably connected to TLB 232 that is associated with system host 120. This μTLB is likewise controlled by one of the megacell 100 processors.

Memory Management Unit

[08] At the megacell traffic controller level, all addresses are physical. They have been translated from virtual to physical at the processor sub-system level by a memory management unit (MMU) associated with each core, such as DSP core 105 and MPU core 103. At the processor level, access permission, supplied through MMU page descriptors, is also checked, while at the megacell level protection between processors is enforced by others means, which will be described in more detail later.

[09] The TLB caches contain entries for virtual-to-physical address translation and access permission checking. If the TLB contains a translated entry for the virtual address, the access control logic determines whether the access is permitted. If access is permitted, the MMU generates the appropriate physical address corresponding to the virtual address. If access is not permitted, the MMU sends an abort signal via signal group 244 to MPU 102.

[10] Upon a TLB miss, i.e., the TLB does not contain an entry corresponding to the virtual address requested, translation table walk software retrieves the translation and access permission information from a translation table in physical memory. Once retrieved, the page or section descriptor is stored into the TLB at a selected victim location. Because a "load and store multiple" instruction may potentially cross a page boundary, the permission access is checked for each sequential address.

[11] Unpredictable behavior will occur if two TLB entries correspond to overlapping areas of memory in the virtual space. This can occur if the TLB is not flushed after the memory is re-mapped with different-sized pages leaving an old mapping with different sizes in the TLB, and making a new mapping that gets loaded into a different TLB location, for example.

MMU/TLB Control Operation

[12] Figure 3 is a block diagram illustrating a shared translation look-aside buffer (TLB) 300 and several associated micro-TLBs (μTLB) 310(0)-310(n) included in megacell 100 of Figure 2. On a μTLB miss, the shared TLB is first searched. TLB controller 320 is alerted by asserting a μTLB miss signal 324. In case of a hit on the shared TLB, the μTLB that missed is loaded with the entry content of the shared TLB 300. In case of miss in shared TLB 300, the shared TLB alerts TLB controller 320 by asserting a TLB miss signal 326. Controller 320 then asserts an interrupt request signal 328 to system interrupt controller 250. Interrupt controller 250 asserts an interrupt to the processor whose OS supervises the resource which caused the miss. A TLB entry register 330 associated with TLB controller 320 is loaded by a software TLB handler in response to the interrupt. Once loaded, the contents of TLB entry register 330 are transferred to both shared TLB 300 and the requesting μTLB at a selected victim location as indicated by arcs 332 and 334.

Shared Cache and RAM

[13] Referring again to Figure 1, Megacell 100 includes large shared memory subsystem 112 that function as a secondary level of RAM (L2 RAM) 113 and cache (L2 Cache) 114. This level of memory is preferably called the outer level, as each processor in various embodiments may have multilevel internal memory. However, for the present embodiment, processors 102, 104 have one level of internal memory, which is referred to herein as L1 within the memory hierarchy, therefore the outer level memory subsystem will be referred to as level two (L2). The

megacell outer memory 112 is organized as what's called a SmartCache, which is a configurable cache and which allows concurrent accesses on cache and RAM-set. RAM-set is a block of RAM that has aspects of cache behavior and cache control operations as well as DMA capability. The SmartCache architecture provides predictable behavior and enhanced real-time performance while keeping high flexibility and ease of use. A detailed description of a SmartCache is provided in US Patent Application Serial No. 09/591,537, (TI-29884) entitled *Smart Cache*. Advantageously, RAM-set configured as a RAM offers fast memory scratchpad feature.

[14] Megacell "outer" memory 112 can be shared between megacell internal processors and external Host processors or peripherals. RAM usage can also be restricted to the usage of a single processor thanks to the MMU mechanism, described earlier. However, in another embodiment a need might arise in the megacell to add additional physical protection per processor on some part of megacell memory to overwrite the MMU intrinsic protection.

[15] A unified shared cache architecture of this embodiment is a four way set associative cache with segmented lines to reduce system latency. All outer memories are treated as unified instruction/data memory to avoid compiler restrictions such as data in program space or vice-versa. Size of this cache or the degree of associativity is a design choice and may vary in other embodiments of the present invention. General construction of set-associative caches are known and need not be described in detail herein. Typically, L1 caches are 16kbytes or 32 kbytes, and the L2 cache is 128kbytes, 256kbytes or larger, for example. Likewise, the number of associated RAM-sets may vary in other embodiments.

[16] RAM-set control registers, such as control register 531 in cache control circuitry 530 (Figure 4), are memory mapped and therefore also benefit from the protection provided by the MMU. However, this would force operations on cache or any specific RAM-set to be on separate pages for protection reasons. Therefore, a control register is provided in TLB control register set 323 (Figure 3) to configure

how and by which CPU the various parts of megacell memory are controlled. All CPUs can execute operations such as cache flushing or cache cleaning as these operations will be restricted by a resource identifier field located in the TAG area of the cache.

[17] Figure 4 is a block diagram illustrating a representative configurable cache 500 that has a cache representative of L2 cache 114 and a RAM-set representative of shared RAM 113. Configurable cache 500 is also representative of L1 cache 203, 204, 223, and 224 that are included respectively in each of the processor modules 102, 104 of Figure 2; however, in the present embodiment, each L1 cache has only a single segment per line. As discussed above, the configurable cache is composed of a 4-way set-associative cache that includes a TAG Array 502(0-3) and Data array 506(2-5) and one or more additional RAM-sets, in this case data arrays 506(0-1). In the present embodiment, data array 506(1-5) are each 32kbytes, while data array 506(0) is 64kbytes.

[18] During an access request, each TAG array 502(0-3) provides a tag value to a respective comparator 546(0-3) and is compared against a most significant portion of a proffered address 548. A tag value is stored in tag array 502(0-3) according to an index value that corresponds to a least significant address of a proffered address. Thus, for any proffered address, an associated tag may be found on anyone of the four tag arrays. If a tag matches a proffered address, then hit/miss logic 510 asserts a respective hit signal hit-way(2-5) 514. In this embodiment, a resource ID (R-ID) field 520 and a task ID (task-ID) field 522 is also included with each entry in the tag array, along with a set of valid bits VI(1-4). Usage of these fields will be described in more detail later. Prefetch circuitry 516 receives signals 512-514 and forms a request to L3 memory when a miss occurs. For each hit, the requested data is provided via bus 541b to an output port of the cache via cache output buffer 540b. In certain embodiments, an L1 cache may have task_ID and R-ID fields, while in other L1 cache embodiments these fields may be omitted.

[19] The RAM-set also includes valid bit arrays 504(0-1) The RAM-set can be configured as a cache extension or as a block of RAM. When configured as RAM, a loading mechanism is provided by a separate DMA engine to optimize data transfer required by multimedia applications. For each hit in the RAM-set portion of the cache, requested data is provided via bus 541a a second output port of the cache via cache output buffer 540a.

[20] Cache control circuitry 530 includes control registers 531 which are used to configure the configurable cache. Fields in the control register include: RAM_fill_mode, Cache_enable, organization, and Full_RAM_base. The control circuitry is coupled to all of the operational blocks of the configurable cache and allows for dynamic reconfiguration of the configurable cache under control of software.

[21] In the embodiment of Figure 4, the RAM-set has two different sized data arrays, Data array 506(0) is 64kbytes and Data array 506(1) is 32 kbytes; however, other embodiments may specify all RAM-sets with the same size to simplify the hardware logic and the software model.

[22] Each RAM-set has an associated TAG register, referred to as Full Set Tag 508(0-1) containing the base address of the RAM-set and a global valid bit (VG) 509(0-1) in addition to an individual valid bit contained in valid bit arrays 504(0-1), referred to as VI, for each segment of each segmented line in the associated data array. Each segment has also a dirty bit referred to as DI, not shown on this figure but on a later one. In the present embodiment, RAM-set lines have the same size as the cache lines; however, in other embodiments a longer line size can also be used to reduce the number of VI bits. RAM-set base registers are coupled with a logical comparison 542(0-1) on a most significant address portion 544 for each access request.

[23] An organization field in cache control register (RAMset-ctrl[n]) 531 for each RAM-set provides the capability to configure it as a cache extension (RAM-set)

or as a plain RAM. When configured as a plain RAM, the valid bits are ignored. Table 1 explains other fields in this register.

Table 1 – Cache Control Register

Bit[0]	0/1 RAM-set 0 operates as a cache or as a RAM
Bit[1]	0/1 RAM-set 1 operates as a cache or as a RAM
DMA mode bit	When set, block operations operate in DMA mode
Fill Mode	Line by line fill, or complete block fill

[24] For L2 caches, there is another control word that indicates which CPU can configure the RAM-set behavior of each L2 RAM-set. This control word is memory mapped and accessible only to the MPU master. For example: Bit[0] : 0/1 CPU master/DSP master for RAM set 0. A status register (not shown) connected to cache control circuitry 530 provides cache information, including number of RAM-sets, sizes, Cache number of way, and line size.

[25] When configured as a RAM, base address registers 508(0-1) are programmed such that this memory does not overlap with other memories in the system. Note, the base address register and the full set tag register are the same. This memory space is mapped as non-cacheable at the outer level. RAM control logic (address decode) generates a hit equivalent signal, which prevents the outer cache from fetching the missing data/instruction to the external memory. VG bit 509(0-1) acts as an enable/disable. It is set when the base address register is written to and cleared when the RAM is invalidated or disabled.

[26] If the register base address of the RAM is programmed in such a way that the associated memory area overlays with the external memory, coherency is not guaranteed by hardware of this embodiment.

[27] When configured as a cache, hit/miss control circuitry 510 generates hit/miss signals called hit-hit 512 and hit-miss 513 for each RAM-set. A hit-hit is generated when a valid entry of the RAM-set matches the address provided by the core. An entry is valid when both VG and its VI are set. A hit-miss signal is

generated when the base address of the RAM is valid (VG = 1) and matches the most significant portion of an address provided by a processor but the selected entry in the RAM-set has its VI equal to zero.

[28] The hit-miss or hit-hit signal has precedence over the hit way (2-5) signals 524 of the 4-way set-associative cache. This implies that any value loaded previously in the cache that should be in the RAM-set is never selected and will eventually be removed from the cache. However, data can create coherency problem in case of modified data (copy back). Therefore, it is recommended to write back ("clean") or even flush the range of address that will correspond to the RAM-set range of addresses. Other embodiments might not have such precedence defined and instead rely on cache invalidate operations to correctly prepare an address range that will be programmed to reside in a RAM-set, for example.

[29] Figure 5 is a flow chart illustrating operation of the hit/miss logic of the configurable cache of Figure 4. In step 550, an address is received from the processor core in connection with a read operation. If the instruction/data cache is disabled, which is checked in step 552, the instruction/data is retrieved from second level memory in step 554. If the cache is enabled, then if either the high order bits of the address from the processor (ADDR[H]) do not match the high order bits of the starting address 508(n) or the global valid bit 509(n) is set to "0" (step 556), then there is a RAM-set miss. In this case, if there is a cache hit in the 4-way set associative cache in step 558, then the information is retrieved from the 4-way set associative cache is presented to the core processor via cache output buffer 540b. If there is a miss in the 4-way set associative cache, the line is loaded into the 4-way cache from second level memory.

[30] Returning again to step 556, if both the high order bits of the address from the processor (ADDR[H]) match the high order bits of the starting address 508(n) and the global valid bit 509(n) is set to "1", then there is a RAM-set hit at the line corresponding to ADDR[L], and the valid entry bits are used to determine whether it is a hit-hit situation where the requested instruction is present in the

RAM-set and can be presented to the processor, or a hit-miss situation where the requested instruction is mapped to the RAM-set, but the information needs to be loaded into the RAM-set's data array 506(n) from the second level memory. If, in step 564, the individual valid entry bit (VI) 504(n) for the line indicates that the line is valid (VI[ADDR[L]]=1), the instruction is present in the RAM-set and is presented to the processor through the RAM-set's output buffer 540a. If, on the other hand, the valid entry bit for the line indicates that the line is not valid (VI[ADDR[L]]=0), the line is loaded into the data array 506(n) of the RAM-set from main memory in step 568.

[31] Figure 6 is an illustration of loading a single line into the RAM-set of Figure 4, in which only one data array 506(0) and its associated bases address register 508(0), global valid bit 509(0) and individual valid bit array 504(0) are illustrated. The RAM-set can be loaded in two ways: Line-by-line fill, and Complete fill/block fill, as indicated by the RAM_fill_mode field of control register 531.

[32] When a new value is written into full-set TAG register (base address) 508(0), all content of the RAM-set data array associated with that TAG register is invalidated by setting individual valid bits 504(0) to logical 0; however, global valid bit 509(0) is set to logical 1. Following the programming of the base address register, the RAM-set will begin to fill itself one line at a time on every hit-miss located in the RAM-set, as discussed with reference to Figure 5. For example, after a miss at an address location corresponding to line 611, data is accessed from second level memory and placed in line 611, VI bit 610 is set to logical 1, and the requested data is provided to the processor.

[33] On the other hand, if a set fill (RAM_fill_mode) is chosen, when the starting address is written to the Full_set_tag register 508(0), all or a portion of the associated data array 506(0) is filled through a block fill process. As each line is loaded from second level memory, the individual valid entry bit 504(0) corresponding to the line is set to "1".

[34] Figure 7 is an illustration of loading a block of lines into the RAM-set of Figure 4. The block fill is based on two additional registers called Start (CNT) 700 and End 702. Start is a 32-n-bit counter and End is a 32-n-bit register, where 2^n represent the number of byte per line. An array area 710 to be filled is defined by an initial value of Start 700a, indicated at 711, and the value of End 702, indicated at 712, for example. In this embodiment, a single block operation can span one or more RAM-set, for example.

[35] Writing a value in End register 702 sets the RAM-set control 530 in block fill mode for the block loading. Setting Start 700 after setting End 702 initiates a block transfer. At this time, all of the individual valid bits associated with array area 710 are set to logical 0. Setting Start address 700 without previously setting the end address or writing the same value in start and end simply loads the corresponding entry. A finite state machine (FSM) represented by flip-flop 720 controls the block fill. FSM 720 is part of control circuitry 530.

[36] Asserting signal 721 causes load signal LD to be asserted to load Start register 700 and initiates the block fill. Signal LD is asserted in response to signal 721 if state machine 720 isn't already performing a block load from a prior command. Signal 721 is asserted in response to specific load operation command or a miss on load, which will be described later. As each line is loaded into array area 710, a corresponding individual valid bit is set to logical 1, such as bit 713, for example. Signal 722 is asserted when counter 700 has been incremented to equal the value in End 702. Signal 723 drives status bit 31 of a SmartCache status register to indicate when a block fill is in operation.

[37] If state machine 720 is already performing a block load, a second one stops the current block load transfer. The system relies on the CPU to check that no active block load operation is on-going if the first prefetch must complete before another is initiated. Another embodiment could signal an error to the CPU or stall the CPU until completion of the current block load. However, the last embodiment

is not suitable for real time system as the stall period becomes highly dependent on the block load size operation.

[38] In the case of multiple RAM-sets, the start address determines in which RAM-set the block load is directed. The selection of the RAM-set is done by comparing the top part of the start address with the contents of the RAM-set base address and loading the bottom part in the counter (CNT). If the start address is not included inside any of the RAM-set, the instruction behaves like a prefetch block or respectively as a prefetch-line on the cache. Depending on the End and Start values, the block size can vary from one line to n lines.

[39] As discussed earlier, the RAM-set of the Configurable cache can be managed in chunks of contiguous memory. Standard cache operations such as miss resulting from a CPU read access on the RAM-set prefetch I/D entry or clean entry are respectively changed into a block prefetch operation or a block cleaning operation if the end of block register 702 has been previously programmed. A block operation can also result from the programming end-of-block register 702 and start-of-block register 700. Clean operations are blocking, but interruptible on the completion of a line in order to guarantee maximum latency for real-time systems. An interrupt stops the block operation to let the CPU process the interrupt and the software then re-starts the block operation when the interrupt return occurs.

[40] The block prefetch operation of the present embodiment re-uses the existing hardware used for full cleaning of the cache; however another embodiment can have a different counter and state machine controller, for example. During the block operation the CPU can be in wait and its activity is resumed on reception of an interruption which stops the current block operation or the CPU can be concurrently running with a single cycle stall during line transfer in the write/read buffer.

[41] Figure 8 is an illustration of interrupting a block load of the RAM-set according to Figure 7 in order to load a single line within the block. To reduce system latency, a megacell processor, referred to generically as a CPU,

advantageously can still access both cache and RAM-set when block loading is in progress; therefore, the following can happen:

[42] (1) The CPU accesses a line already loaded. The CPU is served immediately or after one cycle stall if there is a conflict with a line load.

[43] (2) The CPU accesses a line not yet loaded, referred to as a hit-miss. The CPU is served after the completion of the on-going line load. For example, if an access is made to line 732 prior to being loaded by a pending block load, then VI bit 733 will be logical 0. This will cause the hit-miss signal associated with this RAM-set to be asserted. Line 732 will then be accessed and loaded into data array 730 and the CPU request is satisfied.

[44] In order to take further advantage of the fact that a line within data array 730 has been fetched in response to a CPU access request, each line load is done in two indivisible steps. First, the entry's VI bit is checked by detection circuitry 510 in response to control circuitry 530 to determine if the entry has already been fetched. Then, only if the line is not already present in the cache or in the RAM-set, it is loaded from secondary memory.

[45] Before initiating a block load by programming new values in End and Start, the status must be checked to see that no previous block load is on-going. In this embodiment, there is no automatic hardware CPU stall on this case and doing so would cause the on-going block load to stop. This could result in an unexpected long latency in a real-time applications for accesses into the RAM-set in which the block load was interrupted in this manner. However, in another embodiment, means are provided to allow a second prefetch block command to stop a current active one. Once the second block command is completed, the first one is resumed.

[46] Thus, the present embodiment provides an interruptible prefetch/save block on RAM-set using current cache mechanism: miss on load and prefetch D-line/prefetch I-line respectively for data/instruction after programming the end-of-block register, the CPU being in wait during block operation. Similarly, the present embodiment provides an interruptible clean block operation on RAM set using

current cache mechanism clean-entry after programming the end-of-block register, the CPU being in wait during block operation. For prefetch block, the preferred embodiment is a non blocking operation on the current embodiment.

[47] The present embodiment provides the ability to prefetch block on RAM-set using the cache mechanism: prefetch D-line/ prefetch I-line respectively for data/instruction after programming the end-of-block register with concurrent CPU cache and/or RAM-set access.

[48] The present embodiment performs both of the above using an end-of block register and a start-of block register to initiate block operation (initial value of the block counter).

[49] The present embodiment also extends the Interruptible Prefetch/save block scheme to the cache with no boundary limit between cache and RAM-set. This is the same as cache operation based on range of addresses.

Cache Features

[50] The unified cache memory of the present embodiment supports write back, and write through with/without write-allocate on a page basis. These controls are part of the MMU attributes. Hit under miss is supported to reduce conflicts between requesters and consequent latency. Concurrent accesses on RAM-sets and cache are supported.

[51] Referring again to Figure 4, on a cache miss, the segment corresponding to the miss is fetched from external memory first. For this discussion, data array 506(0) will be discussed, although it is actually configured as a RAM-set instead of Cache. All of the data arrays 506(0-5) have the same organization. Each data array has a number of lines, line 507 being representative, which are segmented into four segments 507(0-3) that each hold 16 bytes data or instruction. For example, in L1 cache 224 if a miss occurs in second segment 507(1), the second segment is fetched from second level RAM 113 or cache 114 or from third level memory 132, 134 if the second level misses. Then, the third segment and finally the

fourth segment are loaded into segments 507(2) and 507(3) automatically, referred to as automatic hardware prefetch. In this embodiment, first segment 507(0) is not loaded into the cache. This sequence of loads can be interrupted on a segment boundary by a miss caused by a request having higher priority. The interrupted load is not resumed, as the remaining segments will be loaded if required later in response to a new miss.

[52] Likewise, second level cache 114 has a data array with a number of lines that are segmented into four segments that each hold 16 bytes. If second level cache 114 misses, it will be filled from third level memory 132, 134 using a multi-cycle operation in which each segment of a given line is accessed. Multi-cycle operations on second level cache 114 are non-blocking. A Multi-cycle cache operation is launched and a status bit indicates its completion. As operations can be initiated by several requesters, such as DSP 104 and MPU 102, these operations can not be blocking due to real time constraints. If one processor initiates a clean_all_task_ID or a block operation for example, other requests can interleave.

[53] Each cache segment has a valid bit (VI) and a dirty bit (not shown) in tag array 502(0-3). Each line such as 507 also has an associated shared bit (not shown) in the tag array. On a write back when a line is replaced, only the segments with modified (dirty) data are written back. Each RAM-set segment has a valid bit (VI) in tag array 504(0-1).

[54] In this embodiment, RAM-sets do not have Task_ID and R-ID fields and shared bit markers associated with each line. Operations on task_ID, R-ID, data marked as shared are limited to the cache. However, another embodiment may harmonize the RAM-set and cache. The hit logic of second level cache 114 only uses the address field. Task-Id and R-Id are used in task operations only.

[55] In this embodiment, L1 caches 202, 206, 222, 226 and L2 cache 114 are organized as 4-way set associative caches. A random cache replacement strategy has been chosen for the replacement algorithm of the 4-way set associative caches.

In this embodiment, the caches do not support cache entry locking except through the RAM-set.

[56] Table 2 includes a listing of the various cache and RAM control operations that can be invoked by the processors in the megacell of the present embodiment. In this embodiment, all operations on an entry operate on segments; there are four segments per entry in the L2 cache, as discussed above. When applied to L1 caches which are segregated into a data cache and a separate instruction cache, then the flush, clean and prefetch operations are directed to the type of information contained in the targeted cache. This means that a way is provided to identify on which cache, instruction or data, a command such as flush applies.

[57] A state machine in cache controller circuitry 530 executes a requested control operation, as indicated by a control word.

[58] In another embodiment, the control operations can be invoked by executing an instruction that invokes a hardware or software trap response. As part of this trap response, a sequence of instructions can be executed or a control word can be written to selected address, for example. In another embodiment, one of the processors may include instruction decoding and an internal state machine(s) to perform a TLB or Cache control operation in response to executing certain instructions which may include parameters to specify the requested operation.

Table 2 – Cache and RAM Control Operations

(C: operation on the cache, RS: operation on RAM-set, R: operation on RAM)

Function		Software view (memory mapped/ co-proc)
Flush_entry (address)	C/RS	Flush the entry[1], whose address matches the provided address or a Range of addresses, if End has been set previously. Flush-range instruction is made of two consecutive instructions Set_End_addr(address) + Flush_entry (address).
Flush_all_entry_of_ task_ID(task_ID)	C	Flush all entries matching to the current taskID in the cache but not in the RAM-set
Flush_all_entry_of_R_ID(tas k_ID)	C	Flush all entries matching to the current R_ID in the cache but not in the RAM-set
Flush_all	C	Flush all entries in the cache but not in RAM-set
Flush_all_shared	C	Flush all entries marked as shared
Flush_all_task_ID_ shared(task_ID)	C	Flush all entries matching the current taskID and marked as shared
Flush_all_task_ID_not_ shared (task_ID)	C	Flush all entries matching the current taskID and marked as not shared
Clean_entry (address)	C/RS	Clean the entry[1], whose address matches the provided address or a Range of address if End has been set previously. Clean-range instruction is made of two consecutive instructions Set_End_addr(address) + Clean_entry (address).
Clean_all_entry_of_ taskID(task_ID)	C	Clean all entries matching to the current taskID in the cache but not in the RAM-set
Clean_all_entry_0f_R_ ID(task_ID)	C	Clean all entries matching to the current R_ID in the cache but not in the RAM-set
Clean_all	C	Clean all entries in the cache but not in RAM-set
Clean_all_shared	C	Clean entries marked as shared
Flush_all_task_ID_ shared(task_ID)	C	Flush all entries matching the current taskID and marked as shared
Clean_all_taskID_not_ shared(Task_ID)	C	Clean all entries matching the current taskID and marked as not shared
Clean&Flush_single_ entry(address)	C/RS	Clean and flush the entry[1], whose address matches the provided address or a Range of address if End has been set previously. Clean-range instruction is made of two consecutive instructions Set_End_addr(address) + Clean_entry (address).
Clean&flush_all_ entry_of_taskID (Task_ID)	C	Clean and flush all entries matching to the current taskID in the cache but not in the RAM-set
Clean&flush_all_entry_ of_R_ID (Task_ID)	C	Clean and flush all entries matching to the current R_ID in the cache but not in the RAM-set
Clean&flush_all	C	Clean and flush all entries in the cache but not in RAM-set
Clean&flush_all_ shared	C	Clean and flush entries marked as shared
Clean&flush_all_ taskID_shared (task_ID)	C	Clean and flush all entries matching the current taskID and marked as shared
Clean&flush_all_taskID_ not_shared (task_ID)	C	Clean and flush all entries matching the current taskID and marked as not shared
Set_RAM_Set_Base_ addr(RAM-setID)	RS/R	Set new RAM-set base address, set VG and clear all VI and set End to last RAM-set address by default preparing the full RAM-set loading. In that case no need to write the END address before writing the start address to load the RAM-set
Set_End_Addr (address)	C/RS	Set end address of the next block load and set the RAM-set controller in block fill mode.
Set_start_addr (address)	C/RS	Set start address of a block and initiates the loading of this block
Prefetch-entry(address	C/RS	Prefetch-the entry, whose address matches the provided address or a Range of address if End has been set previously. Prefetch-range instruction is made of two consecutive instructions Set_End_addr(address) + Prefetch_entry (address).
Flush_RAM-set (RAMset_ID)	RS/R	Clear VG and all VI of the selected RAM-set

Detailed Aspects

[59] Various aspects of the digital system of Figure 1 will now be described in more detail.

[60] Figure 9 is a flow diagram illustrating an interruptible block operation on the memory circuitry of Figure 4. As discussed earlier (see Table 2 – Cache and RAM Control Operations), a block operation can be performed to load or to clean a portion of the cache or RAM-set. This discussion will describe a block load, but a block clean operates in a similar manner. In step 900, a block operation is initiated to load a selected portion of the segments in the cache or RAM-set, according to a value stored in end register 702 and in start register 700. The operation is initiated by writing an operation directive, as listed in Table 2, to control circuitry 530 along with a starting address that is loaded in start register 700. In this embodiment, all segments within the selected block of segments are automatically invalidated in response to initiating the block operation. However, in another embodiment, segments may or may not be automatically invalidated.

[61] In step 902, prior to loading a line, the valid bit (VI bit 504(0), see Figure 7) associated with the line, or with the segment of the line if there are multiple segments per line, is tested to determine if the segment contains valid data. If the segment does not contain valid data, then a line or a segment is fetched in step 904 and the segment is then marked as being valid by setting a corresponding valid bit. On the other hand, if step 902 determines that a segment contains valid data from a prior data transfer operation, then step 904 is inhibited and a transfer to the valid segment is not performed, as illustrated by link 903. Advantageously, performance is improved by inhibiting transfers to segments that already have valid data.

[62] In step 906, a test of end register 702 is made to determine if the end of the block has been reached. If so, the block operation is completed at step 910. If the end of the block has not been reached, then the next address is selected in step 908 and steps 902, 904 and 906 are repeated.

[63] In this embodiment of the present invention a processor connected to the memory circuit can continue to execute instructions during a block operation. In so doing, it may access an address during step 920 that is within a block that is being loaded as a block operation, as discussed with respect to Figure 8.

[64] In step 922 miss circuitry checks a valid bit associated with the segment that is accessed by the processor in step 920 to determine if the segment contains valid a instruction or data value. If the CPU accesses a line already loaded, the CPU is served immediately or after one cycle stall (conflict with a line load), as indicated by arc 923.

[65] If the CPU accesses a line not yet loaded (hit-miss), then the CPU is served after the completion of an on-going block line load. For example, if an access is made to line 732 prior to being loaded by a pending block load, then VI bit 733 will be logical 0. This will cause the hit-miss signal associated with this RAM-set to be asserted in step 922. Line 732 will then be accessed and loaded into data array 730 in step 926 and the CPU request is satisfied. During step 926, the valid bit for the segment just fetched is asserted, so that a later access attempt by the block loading circuitry will be inhibited, as discussed in step 902.

[66] In step 924, this embodiment of the memory circuit performs a test after a miss is detected in step 922. If end register 702 has been loaded with an end-of-block address and a block transfer is not currently underway, as indicated by status signal 723 from FSM 720, then a block operation is commenced, as indicated by arc 925. In this case, the block operation starts at an address provided by the miss detection circuitry associated with the miss detected in step 922. The block operation ends at the address provided by the end register.

[67] Another embodiment of the memory circuit may not provide the feature illustrated by step 924. Other combinations of the various features illustrated in Figure 9 may be provided in other embodiments of the memory circuit. For example, another embodiment may provide block initiation by step 924, but not provide an operation initiation step 900.

[68] Figure 10 is a block diagram of the level two (L2) cache of Figure 7 illustrating data flow for interruptible block prefetch and clean functions in the RAM-set portion. As discussed earlier, each segment has an associated individual valid bit (VI) and an individual dirty bit (DI). The VI bit indicates the associated segment contains valid data. Valid data is placed in a segment in response to a fetch after a miss, or a preemptive block load. In some embodiments, a DMA transfer can also place valid data in a segment. The DI bit indicates the data in the associated segment is "dirty," meaning it has been changed in some manner that is not reflected in the secondary memory location from which it was originally fetched. Generally, this is due to a write transaction by CPU 1500. In another embodiment, a DMA transaction can place dirty data into the cache. In this embodiment, there are four segments per line, therefore there are four valid bits and four dirty bits per line. During a clean operation, a dirty line is first held in write buffer 1504 pending transfer to external memory 1502. Memory 1502 is representative of external memory 132 or on chip external memory 134 (Figure 1). During a block load operation, a data line is transferred from external memory 1502 to data array 710 under control of FSM 720, as described earlier.

[69] Figure 11 illustrates operation of the cache of Figure 4 in which a block of lines is cleaned or flushed in the set associative portion. Programming register "end of block" 702 changes a cache operation such as clean or flush for a single specified entry to an operation on a block of lines located between this specified entry and the entry pointed by "end of block" register 702. The function can also be implemented using "end-of block" register 702 and start-of block register 700 to hold an initial value of the block counter. Finite state machine 720 controls the cache block flush and clean operations, as described previously with respect to Figure 7 and Figure 8 for cleaning and flushing the RAM-set. In the present embodiment, the same FSM and address registers are used to control cache cleaning and RAM-set cleaning operations.

[70] Thus, a cache clean and/or a cache flush operation can be performed on a range of addresses in response to a software directive.

[71] In another embodiment, separate control circuitry can be provided for the cache and for the RAM-set. Alternatively, in another embodiment a RAM-set may not be included.

[72] Figure 12 is a flow diagram illustrating cleaning of a line or block of lines of the cache of Figure 4. As discussed earlier (see Table 2 – Cache and RAM Control Operations), a block operation can be performed to load or to clean a portion of the cache or RAM-set. This discussion will describe a block clean operation, but a flush can be performed in a similar manner. In step 1200, an operation is initiated to clean a selected portion of the segments in the cache or RAM-set, according to a value stored in end register 702 and in start register 700. The operation is initiated by writing an operation directive, as listed in Table 2, to control circuitry 530 along with a starting address that is loaded in start register 700. However, if a block operation is to be performed instead of a single line operation, then an ending address value is written into end register 702 prior to loading the start register.

[73] In step 1202, this embodiment performs a test after an operation is initiated in step 1200. If end register 702 has not been loaded with an end of block address, then step 1204 checks a dirty bit associated with the address selected by start address register. As mentioned earlier, this embodiment has four dirty bits for each line. The start register and end register contain addresses that are line aligned, so all four dirty bits on each line are checked. If the dirty bit indicates the associated line contains dirty data, then the segments which have dirty data are written to secondary memory in step 1206. If the line does not contain dirty data, then the operation is completed as indicated in step 1210.

[74] Another embodiment may not provide the feature illustrated by step 1202 and instead only provide block operations or only provide single line operations, for example.

[75] Referring back to step 1202, if end register 702 has been loaded with an end-of-block address and a block transfer is not currently underway, as indicated by status signal 723 from FSM 720, then a block operation is commenced, as indicated by arc 1203. In this case, the block operation starts at an address provided by start address register 700. The block operation ends at the address provided by the end register.

[76] In step 1214 a check is made of a dirty bit associated with the address selected by start address register, a described for step 1204. If the dirty bit(s) indicates the associated line contains dirty data, then that line or segments are written to secondary memory in step 1216. If the line does not contain dirty data, then a write transaction is not required.

[77] In step 1218, a test of end register 702 is made to determine if the end of the block has been reached. If so, the block operation is completed at step 1210. If the end of the block has not been reached, then the next address is selected in step 1220 by incrementing the start register and steps 1214, 1216, and 1218 are repeated.

[78] In this embodiment a processor connected to the memory circuit can continue to execute instructions during a block operation. In so doing, it may access an address that is within a block that is being cleaned as a block operation, as discussed with respect to Figure 8. During the block operation the CPU can be in wait and its activity is resumed on reception of an interruption which stops the current block operation or the CPU can be concurrently running with a single cycle stall during line transfer in the write/read buffer.

[79] A flush operation is performed in a similar manner. A flush simply invalidates each valid bit within the range selected by the start register and the end register. The same flow is used. In steps 1204 and 1214, the selected valid bit(s) is checked. If it is in a valid state, it is reset to an invalid state in steps 1206, 1216.

[80] A clean and flush operation is also performed using the same flow. In this case. in steps 1204 and 1214, the selected valid bit(s) and the selected dirty

bit(s) are checked. If a valid bit is in a valid state, it reset to an invalid state in steps 1206, 1216. If a dirty bit is asserted, the associated segment is written to secondary memory in step 1206, 1216.

[81] Figure 13A is a block diagram of an embodiment of a local memory similar to the RAM-set memory of Figure 7 illustrating an embodiment of the present invention. This is an embodiment of a local memory 2003 that can be provided in place of or in addition to data cache 203 or 223, for example, (refer to Figure 2A and Figure 2B). Local memory 2003 uses of a set of indicator bits 2020 to support concurrent CPU and DMA access, indicated at 2040 and 2041, respectively. Local memory 2003 is not a cache in that it does not overlay a portion of secondary memory, but is instead an independent portion of memory that occupies it's own portion of the address space of processor 2000. The local memory has a data array 2006 that is segmented in lines with individual valid bits VI enabling CPU 2000 to access any line outside or inside an active DMA range concurrently while the DMA transfer is on going.

[82] DMA circuitry 2030 includes a finite state machine (FSM) 2020, start register 2022 and end register 2024. In this embodiment, the Configurable cache commands (Table 2) are also used to initiate DMA transfers; however, other embodiments may have specific DMA commands. Source/destination register 2010 provides a destination/source address that enables transfer of data or instructions from an address space associated with data array 2006 to a different address space during transfer from/to external memory 2002.

[83] Writing a value in End register 2024 sets DMA circuit 2030 in block fill mode for block loading. Setting Start 2022 after setting End 2024 initiates a block transfer. At this time, all of the individual valid bits associated with array area 2060 are set to logical 0. Setting Start address 2022 without previously setting the end address or writing the same value in start and end simply loads the corresponding entry. A finite state machine (FSM) represented by flip-flop 2020

controls the block fill. During DMA transfers for either a single prefetch entry or a block transfer, CPU 2000 continues running.

[84] Figure 13B is a schematic for an address calculation circuit for the DMA system of Figure 13A that is connected to modify source/destination register 2010. If CPU 2000 is accessing a line in local memory 2003 that has not yet been loaded by the DMA, a miss is indicated by valid bits VI. In response to the miss, the DMA operation is momentary stalled to load the line requested by the CPU and the DMA operation is then resumed. In order to guarantee real-time performance during this stall, an interrupt will take precedence over the miss process. This sequence is similar to the operation of a RAM-set as described with reference to Figure 7 and Figure 8, except here the address used to access the requested line must be modified so that it points to the correct location in the selectable region 2050 of memory 2002. This address is calculated from both the current DMA internal RAM pointer 2022 (CNT) and the DMA current external pointer 2010 (CNT-EXT). This is easily done by calculating the offset from the current DMA internal RAM pointer 2022 with a simple XOR/OR logic circuit 2026 and an adder/sub 2028 connected to Src/dest register 2010.

[85] For example, CPU 2000 requests a transaction by providing an address on bus 1644. If the requested address falls within an active DMA region, as indicated at 1644a, and the miss signal is asserted. The DMA transfer is halted while the requested line is fetched from region 2050 of memory block 2002. The address for this access is formed by comparing address 1644b provided by CPU 2000 to the value 2022a in count register 2022 in order to form an offset value. The offset value is formed by bit-wise XOR circuit 2026 using the two values to form a result 2027a, determining the leading 1 of this result, as indicated at 2027c and inverting all bits that are less significant to form offset 2027b, a value equal to the difference minus one. The offset is provided to adder 2028 via signal 2027d and added to the contents of destination register 2010 plus one to form an address that is stored into destination register 2010. Destination register 2010 now points to the requested

location, indicated at 2010b. Once the requested line is transferred to local RAM 2003, the offset value on signals 2027d plus one is subtracted from the value in destination register 2010 to restore the original value, indicated at 2010a. The DMA transfer is now resumed.

[86] Other embodiments may provide alternate ways of calculating the requested address using other combinations of registers and adders, for example.

[87] Similar to the RAM-set operation described with respect to Figure 9, prior to loading each line, DMA engine 2030 checks a valid bit associated with that line from the set of valid bits 2020 to avoid overwriting a valid line, which would have been loaded ahead of the DMA execution in response to a CPU access.

[88] In this embodiment, the VI bits are asserted by DMA transfers. In another embodiment, an additional DMA-VI mode bit is provided in a control register associated with the DMA. When set to a DMA-VI mode, each DMA transfer of data into data array 2006 causes the associated VI bit to be asserted. When not in DMA-VI mode, the associated VI bits are not asserted by a DMA transfer.

[89] Advantageously, when the associated valid bits VI of the local memory 2003 are asserted in response to a DMA transfer, then they can be used to monitor the DMA progress, as discussed for block transfer with respect to Figure 7 and 8. This allows CPU 2000 to have access to local memory 2006 concurrently with the DMA operation, including within the range of addresses that are being transferred by the DMA operation.

[90] Referring again to Figure 13A, in this embodiment of local memory 2003, a set of dirty bits are also provided as part of indicator bits 2020. Each line segment of local memory 2006 has an associated dirty bit that can indicate when data in an associated segment has been modified by either processor 2000 or by a DMA transfer.

[91] A base register 2008 is included with local memory 2003 that can be programmed to position the local memory within the address space of processor 2000. A valid bit VG indicates when the base register contains a valid base address.

In other embodiments, the address position of the local memory may be fixed, such that base register 2008 and valid bit VG are not provided.

[92] In this embodiment. all or a portion of the VI bits can be forced to an asserted state under control of software so that the miss detection circuitry always indicates that the associated locations contain valid data in the course of its normal operation. This effectively disables the miss detection circuitry so that the memory operates as a simple local RAM. In this mode, DMA transfers are still available.

[93] Figure 14 is a schematic illustration of operation of an alternative embodiment of the local memory of Figure 13A. As before, miss detection circuitry 510a determines when a location in local memory 2006 is accessed by comparing a most significant portion 1644a of a transaction request address 1644 from CPU 2000 to a value stored in base address register 2008. Miss detection circuitry 510a determines when a location in local memory 2006 contains valid data by determining if both base register valid bit VG and an individual valid bit VI associated with the segment selected by a least significant portion 1644b of the transaction address are both asserted. If so, the transaction request is satisfied by transferring a data value from local memory 2006 to the CPU, as indicated by arc 2040. If VI is not asserted, this indicates that the accessed address is in the range of an on going DMA load.

[94] The local memory can be operated in a manner such that when a transfer request from the processor requests a segment location in the local memory that does not hold valid data, valid data is transferred from selectable location 2050 in secondary memory, as defined by source/destination register 2010, after offset address calculation as discussed above. This can be a single segment transfer, or a block prefetch transfer, as described earlier.

[95] When a miss occurs, processor 2000 is stalled while the segment is being fetched. In an embodiment that does not provide single segment transfer, this stall time may become unacceptable. In this embodiment, in order to prevent response time errors, a timer circuit 2090 is provided. If a stall exceeds a

predetermined time limit, the processor is interrupted by interrupt signal 2091. The time limit can be fixed, or can be programmed by the processor, for example.

[96] In this embodiment of the invention, there is a miss mode latch 1670 that can be loaded by CPU 1600. When set to miss-enable mode, the output of the latch is not asserted and OR gate 1672 passes the Hit signal without effect. However, when set to miss-disable mode, the output is asserted and the hit signal from OR gate 1672 is always asserted. This effectively disables the miss detection circuitry 510a so that the memory 2006 operates as a simple local RAM. In this mode, DMA transfers are still available.

[97] Other embodiments may use a different means to effectively disable the miss circuitry and thereby evoke RAM type operation. For example, all or a portion of the VI bits can be forced to an asserted state so that the miss detection circuitry indicates that the associated locations contain valid data in the course of its normal operation. This could be done after completion of a DMA block operation in order to allow the data obtained by a DMA transfer to be treated from then on as valid data.

[98] Likewise, a portion of the VI bits can be set to an asserted state under control of software so that the miss detection circuitry will not indicate a miss; the associated portion of the local memory will then operate as a simple local memory and can be written to and then accessed by processor 1600.

[99] Figure 15 is a schematic illustration of operation of the indicator bits of the local memory of Figure 13A. In this embodiment, a reset of processor 2000 causes all individual valid bits VI to be set to an asserted state (1), by default. Advantageously, for a system which requires only a tightly coupled local RAM, boot code simply needs to initialize base register 2008 with an address pointing to non existing external memory, as indicated by 2003, such that memory array 2006 does not overlay the address space of any other memory. When base register 2008 is programmed, base register valid bit VG is asserted (set to 1) to enable operation of the memory, as described earlier. The local memory then behaves as simple RAM

memory; since all of the valid bits are asserted, the miss detection circuitry never detects a miss.

[100] Figure 16A and Figure 16B are schematic illustrations of operation of the local memory of Figure 13A with mode circuitry controlling DMA transfers according to dirty bits. In this embodiment, when the DMA circuitry transfers a block of data from a portion 2060 of local memory 2006 to a selectable location 2050 in secondary memory 2002, the dirty bits control which segments get transferred, depending on the state of full transfer (FT) mode bit 2081. FT bit 2081, defined in Table 3, is implemented as a bit in a DMA control register that is programmable by CPU 2000.

Table 3 – Control bit for DMA transfers

Transfer Mode (FT)	When not set, DMA transfers only dirty data. When set, DMA transfers a block of data without regard to dirty bits

[101] Regardless of the state of FT bit 2081, when a DMA transfer from a selectable region in secondary memory to a portion of local memory is performed, the dirty bits associated with this portion of local memory are all reset to indicate clean data and the valid bits associated with this portion of local memory are set to indicate valid data is present.

[102] When CPU 2000 performs a transaction that stores data into a location in the local memory, the dirty bit corresponding to that location in local memory is set indicating that location now holds data that is dirty, i.e., the data in local memory is not coherent with the secondary memory. In this case, only data items within this block of segments that have been modified by processor 2000 after the DMA transfer will be set dirty because the dirty bits were cleared by the DMA transfer.

[103] Referring to Figure 16A, if FT bit 2081 is set to a first state, such as logical 0, a full DMA transfer is not performed on selected block 2060. If a segment 2061 is marked as being dirty, such as bit 2085, then it is transferred as indicated

by arc 2052; however, if a segment is marked as being clean (dirty bit off), then it is not transferred to secondary memory by the DMA circuitry.

[104] Referring to Figure 16B, if FT bit 1681 is set to a second state, such as logical 1, a full DMA transfer is performed. In this case, the DMA circuitry transfers all segments within a selected block 2060 regardless of the state of the corresponding dirty bits 2086.

[105] In this embodiment, there is a dirty bit associated with each segment of the local memory. In alternative embodiments, a single dirty bit may be associated with a set of segments in order to reduce cost, for example.

[106] Figure 17A and Figure 17B are schematic illustrations of operation of the local memory of Figure 13A with mode circuitry controlling setting of the dirty bits. Mode circuitry 2080 controls how dirty bits are set in response to a DMA transfer. DMA dirty-bit enable (DDE) bit 2080 is implemented as a bit in a control register associated with the local memory that can be loaded as needed by software executing on processor 2000. DDE bit 2080 is defined in Table 4.

Table 4 – Control Bit for DMA Transfers

DMA Dirty bit enable	When set, DMA transfers to the local memory cause corresponding dirty bits to be set to a dirty state When not set, DMA transfers to the local memory cause corresponding dirty bit to be reset to a clean state

[107] In Figure 17A, when DDE bit 2080 is set to a first state, such as logical 0, then transfer of a block of data by the DMA controller from secondary memory portion 2050 into a portion 2060 of local memory array 2006 (as indicated by arc 2052) causes the corresponding set of dirty bits 2084a to be set to a "clean" state indicated by "0". The corresponding set of individual valid bits are also asserted so that the miss circuitry will now treat these locations as having valid data. After the DMA transfer, when processor 2000 modifies data in this region of the local memory the corresponding dirty bits will be set. In this case, only data items within this

block of segments that have been modified by processor 2000 after the DMA transfer will be set because the dirty bits were cleared by the DMA transfer.

[108] In Figure 17B, when DDE bit 2080 is set to a second state, such as logical 1, then transfer of a block of data by the DMA controller from secondary memory portion 2050 into a portion 2060 of local memory array 2060 causes the corresponding set of dirty bits 1684b to be set to a "dirty" state indicated by "1". The corresponding set of individual valid bits are also asserted so that the miss circuitry will now treat these locations as having valid data.

[109] Figure 18A and Figure 18B are schematic illustrations of operation of the local memory of Figure 17A and Figure 17B with mode circuitry controlling DMA transfers according to dirty bits. In this embodiment, when the DMA circuitry transfers a block of data from a portion of the local memory 2006 to a selectable location 2050 in secondary memory 2002, the dirty bits control which segments get transferred, depending on the state of full transfer (FT) mode bit 2081. FT bit 2081, defined in Table 3, is also implemented as a bit in the control register associated with the local memory that is programmable by CPU 2000.

[110] Referring to Figure 18A, if FT bit 2081 is set to a first state, such as logical 0, a full DMA transfer is not performed on selected block 2060. If a segment 2061 is marked as being dirty, such as bit 2085, then it is transferred as indicated by arc 2052; however, if a segment is marked as being clean (dirty bit off), then it is not transferred to secondary memory by the DMA circuitry.

[111] Referring to Figure 18B, if FT bit 2081 is set to a second state, such as logical 1, a full DMA transfer is performed. In this case, the DMA circuitry transfers all segments within a selected block 2060 regardless of the state of the corresponding dirty bits 2086.

[112] Advantageously, this combination of DDE bit 2080 and FT bit 2081 allows very flexible interaction using DMA transfers to selectable regions in the secondary memory. For example, one or several small blocks of data could be transferred from various selectable locations in secondary memory when DDE bit

2080 is set so that the dirty bits are turned on; then a larger region could be selected for transferred from the local memory to the secondary memory and advantageously, only the portion(s) corresponding to the one or several small blocks would actually be transferred because of the dirty bits. Many other scenarios are possible.

Digital System Embodiment

[113] Figure 19 illustrates an exemplary implementation of an example of such an integrated circuit in a mobile telecommunications device, such as a mobile telephone with integrated keyboard 12 and display 14. As shown in Figure 19, the digital system 10 with a megacell according to Figure 2 is connected to the keyboard 12, where appropriate via a keyboard adapter (not shown), to the display 14, where appropriate via a display adapter (not shown) and to radio frequency (RF) circuitry 16. The RF circuitry 16 is connected to an aerial 18.

[114] It is contemplated, of course, that many other types of communications systems and computer systems may also benefit from the present invention, particularly those relying on battery power. Examples of such other computer systems include personal digital assistants (PDAs) portable computers, smart phones, web phones, and the like. As power dissipation is also of concern in desktop and line-powered computer systems and micro-controller application, particularly from a reliability standpoint, it is also contemplated that the present invention may also provide benefits to such line-powered systems.

[115] Fabrication of the digital systems disclosed herein involves multiple steps of implanting various amounts of impurities into a semiconductor substrate and diffusing the impurities to selected depths within the substrate to form transistor devices. Masks are formed to control the placement of the impurities. Multiple layers of conductive material and insulative material are deposited and etched to interconnect the various devices. These steps are performed in a clean room environment.

[116] A significant portion of the cost of producing the data processing device involves testing. While in wafer form, individual devices are biased to an operational state and probe tested for basic operational functionality. The wafer is then separated into individual dice which may be sold as bare die or packaged. After packaging, finished parts are biased into an operational state and tested for operational functionality.

[117] The digital systems disclosed herein contain hardware extensions for advanced debugging features. These assist in the development of an application system. Since these capabilities are part of the megacell itself, they are available utilizing only a JTAG interface with extended operating mode extensions. They provide simple, inexpensive, and speed independent access to the core for sophisticated debugging and economical system development, without requiring the costly cabling and access to processor pins required by traditional emulator systems or intruding on system resources.

[118] As used herein, the terms "applied," "connected," and "connection" mean electrically connected, including where additional elements may be in the electrical connection path. "Associated" means a controlling relationship, such as a memory resource that is controlled by an associated port. The terms assert, assertion, de-assert, de-assertion, negate and negation are used to avoid confusion when dealing with a mixture of active high and active low signals. Assert and assertion are used to indicate that a signal is rendered active, or logically true. De-assert, de-assertion, negate, and negation are used to indicate that a signal is rendered inactive, or logically false. References to storing or retrieving data in the cache refer to both data and/or to instructions.

[119] While the invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various other embodiments of the invention will be apparent to persons skilled in the art upon reference to this description. For example, in another embodiment, valid bits may be dispensed with altogether and only dirty bit may be provided, or

visa versa. The local memory may be positioned at a level other than L2. The local memory may have a different set organization with a different number of segments per line, for example. Likewise, the start and end registers may contain addresses that are segment aligned rather than line aligned. References to data being stored in a local memory segment are to be interpreted as meaning data or instructions.

[120] It is therefore contemplated that the appended claims will cover any such modifications of the embodiments as fall within the true scope and spirit of the invention.